**Practical – 1**

**Aim: Write a program to create a client class for blockchain application**

**Code:**

***Cargo.toml***

[package]

name = "simple-blockchain-rs"

version = "0.1.0"

edition = "2021"

authors = ["Utsav Balar"]

# See more keys and their definitions at https://doc.rust-lang.org/cargo/reference/manifest.html

[dependencies]

crypto-hash = "0.3.4"

[dependencies.secp256k1]

version = "0.24.0"

features = ["rand-std", "serde"]

***lib.rs***

pub mod client;

***main.rs***

use simple\_blockchain\_rs::client::Client;

fn main() {

let client = Client::new();

println!("client public key: {}", client.identify());

}

***client.rs***

pub use secp256k1::ecdsa::Signature;

pub use secp256k1::{rand, All, Secp256k1};

pub use std::str::FromStr;

pub mod key {

pub use secp256k1::PublicKey;

pub use secp256k1::SecretKey;

}

/// A client structure that can be used to interact with a blockchain.

///

/// `public\_key` contains the public key of the client.

/// `secret\_key` contains the private key of the client.

/// `secp` contains the secp256k1 context.

pub struct Client {

pub secp: Secp256k1<All>,

pub secret\_key: key::SecretKey,

pub public\_key: key::PublicKey,

}

impl Client {

/// This method creates a new client with a random key pair.

#[allow(deprecated)]

pub fn new() -> Self {

let mut rand = rand::rngs::OsRng {};

let secp = Secp256k1::new();

let (secret\_key, public\_key) = secp.generate\_keypair(&mut rand);

Self {

secp,

secret\_key,

public\_key,

}

}

/// This method identifies the client to the blockchain.

pub fn identify(&self) -> String {

self.public\_key.to\_string()

}

/// This method signs a transaction with the client's private key.

pub fn sign(&self, transaction: &[u8]) -> Signature {

let message = secp256k1::Message::from\_slice(transaction).unwrap();

self.secp.sign\_ecdsa(&message, &self.secret\_key)

}

}

**Client’s public key on stdout**

Text

Description automatically generated